The World Modeler : the nexus between Janus and Battlefield Distributed Simulation-Developmental

Johnson, Matthew A.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

**THE WORLD MODELER:
THE NEXUS BETWEEN JANUS AND BATTLEFIELD
DISTRIBUTED SIMULATION-DEVELOPMENTAL**

by

Matthew A. Johnson

September 1994

Thesis Advisor: David Pratt
Second Reader: Bert Lundy

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1994	Master's Thesis

4. TITLE AND SUBTITLE
THE WORLD MODELER: THE NEXUS BETWEEN JANUS AND BATTLEFIELD DISTRIBUTED SIMULATION-DEVELOPMENTAL (U)

5. FUNDING NUMBERS

6. AUTHOR(S)
Johnson, Matthew A.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The United States Army has two disparate combat models; Janus and Battlefield Distributed Simulation-Developmental (BDS-D). Both facilitate training, tactical development and weapons analysis. The major problem addressed by this research is that entities existing in the Janus Combat Modeler cannot interact with entities in BDS-D. If interaction is made possible, it would produce a synergy between the combined models, each model bringing advantages to the other.

The approach taken was first to identify the differences between the two environments of Janus and BDS-D. Next, a software architecture was developed to store and manipulate data about both simulations. A communications architecture was created to allow data flow between the two environments. Finally, algorithms were developed to allow for interaction between Janus and BDS-D entities.

The result was to integrate Janus, a two dimensional, constructive combat model, into the three dimensional, entity-level virtual battlefield of BDS-D. Janus entities interact in real time with other entities in the BDS-D virtual world. The finished product, the World Modeler, is a software system operating on a low-end Silicon Graphics workstation with TCP/IP and UDP/IP networking capabilities.

14. SUBJECT TERMS	15. NUMBER OF PAGES
DIS, distributed, interactive, simulation, Janus, BDS-D, dead reckoning TCP/IP, UDP/IP,	77
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited

THE WORLD MODELER: THE NEXUS BETWEEN JANUS AND BATTLEFIELD DISTRIBUTED SIMULATION-DEVELOPMENTAL

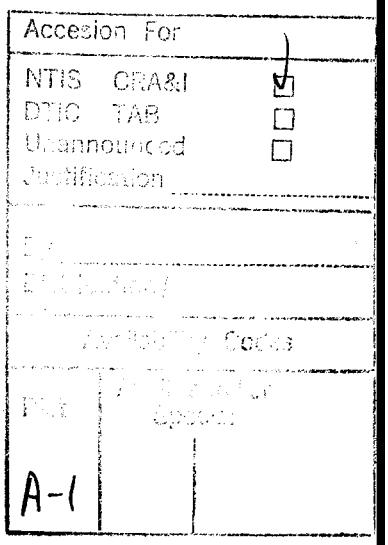
Matthew A. Johnson
Captain, United States Army
B.S., United States Military Academy, 1984

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1994



Author: [redacted]
Matthew A. Johnson

Approved By: [redacted]
David R. Pratt, Thesis Advisor

[redacted]
G. M. Lundy, Second Reader

[redacted]
Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

The United States Army has two disparate combat models; Janus and Battlefield Distributed Simulation-Developmental (BDS-D). Both facilitate training, tactical development and weapons analysis. The major problem addressed by this research is that entities existing in the Janus Combat Modeler cannot interact with entities in BDS-D. If interaction is made possible, it would produce a synergy between the combined models, each model bringing advantages to the other.

The approach taken was first to identify the differences between the two environments of Janus and BDS-D. Next, a software architecture was developed to store and manipulate data about both simulations. A communications architecture was created to allow data flow between the two environments. Finally, algorithms were developed to allow for interaction between Janus and BDS-D entities.

The result was to integrate Janus, a two dimensional, constructive combat model, into the three dimensional, entity-level virtual battlefield of BDS-D. Janus entities interact in real time with other entities in the BDS-D virtual world. The finished product, the World Modeler, is a software system operating on a low-end Silicon Graphics workstation with TCP/IP and UDP/IP networking capabilities.

TABLE OF CONTENTS

I. INTRODUCTION

A. OBJECTIVE

The objective of this research effort was to develop an interface system (the World Modeler) which allows Janus to interact in the Battlefield Distributed Simulation-Developmental environment (BDS-D). The success of this project was measured by the World Modeler's ability to accurately and efficiently integrate Janus entities into BDS-D.

B. BACKGROUND

The Army is preparing for the 21st Century. A program called the Louisiana Maneuvers has been developed as a structured process to investigate and integrate future technologies into the Army. Louisiana Maneuvers is named after a similar effort conducted just prior to World War II. New equipment and tactics were available at that time, but had not been tested in battle. Numerous force-on-force maneuvers were conducted in the Carolinas to test and validate the new technology and tactics before entering the war.

The expense of fielding such forces for the purpose of testing and analyzing future technologies is immense. The Louisiana Maneuvers of today are conducted in Army Battle Labs located throughout the United States; interconnected through Distributed Interactive Simulation (DIS). This connectivity creates a disbursed, three dimensional synthetic environment (BDS-D) which allows for cost effective and timely analysis of future technologies.[ARMY93]

The Computer Science Department at the Naval Postgraduate School has been a principle investigator of the DIS synthetic environment. The department has developed a low-cost, real time, high resolution visual simulator, NPSNET, which is DIS compliant. [PRAT93] With its networking abilities, NPSNET can enter and interact in the BDS-D world. This project utilizes NPSNET as a surrogate crew station in place of a more expensive BDS-D crew station.

C. MOTIVATION

"Louisiana Maneuvers and Battle Labs enable the Army's leaders to see into the future and develop alternatives so that resources, programs, and policies can be adjusted whenever necessary"[ARMY95]. BDS-D is the battlefield to develop and test the Army of the future. However, the need exists to populate the virtual world of BDS-D. Using one computer system to emulate one entity would quickly become cost prohibitive. Attempting to integrate current combat simulations which control large numbers of entities appears to be a cost effective solution this problem. Janus is a viable candidate with its ability to control up to 1200 entities per computer platform.

Advance Technology Demonstrations (ATDs) are being developed and tested in the Battle Labs to evaluate future systems. The Janus - BDS-D connection will provide a "... synthetic simulation environment for high fidelity analysis of division sized engagements in... Louisiana Maneuvers."[Milton]. Deeper analytical insights may be achieved by viewing a Janus battle in the three dimensional world of BDS-D. Additionally, observing how man-in-the-loop simulators effect the outcome of a Janus battle would lend additional insight to the current analysis of Janus scenarios. The World Modeler is the lynch-pin which brings to fruition the synergistic effects of the Janus -BDS-D alliance.

D. ORGANIZATION OF THESIS

This thesis is organized into nine chapters. This chapter provides the background and motivation of the work performed. Chapter II provides an overview of the Janus Combat Modeler, BDS-D, and NPSNET, while an overview of the World Modeler to include its relationship with Janus and BDS-D is in Chapter III. The major functions of the World Modeler: network management, dead reckoning, entity and terrain reconciliation, and event arbitrations are discussed in Chapters IV through VII respectively. Chapter VIII provides information on the run-time performance of the World Modeler with respect to Janus entity integration into BDS-D along with its limitations and applications. Finally, the

Appendices consist of the World Modeler user's guide and the local communications protocol used between Janus and the World Modeler.

II. JANUS, BDS-D, AND NPSNET OVERVIEW

A. JANUS

The Janus Combat Modeler, as described in [JANU93] is a constructive, monolithic combat model used primarily by the United States Army. Its main purpose is two fold. First, it is used as a combat development tool, analyzing both weapon system and tactics. Second, Janus is used as a training tool by leaders for the purpose of tactics training and staff planning.

Janus portrays both entity level systems as well as aggregate level such as platoon or batteries. User's of Janus develop combat scenarios consisting of force-on-force engagements between two opposing forces. Due to the stochastic modeling of entity engagements, executions (referred to as runs) of the same scenario will produce different results. The study of several runs of the same scenario lend to the analysis of the tactics, force structure and weapons systems used. This analysis is used both for training and combat development.

Janus employs high resolution algorithms which accurately model numerous weapons platforms; ranging from a dismounted riflemen to a MLRS battery. This high fidelity of modeling allows Janus to be a valid environment for combat development. Military systems partaking in ground combat is the primary modeling focus of Janus.

Janus is monolithic in nature, operating on one computer system. Its numerous two dimensional displays provide the military expert with a highly detailed, graphic display of the battlefield. User interaction is through a four button mouse called a puck and the two dimensional display.

The Janus model is made up of sixteen FORTRAN executable programs and associated databases. Janus can operate on two platforms. One version uses Digital VAX machines utilizing the VMS operating system and X-Windows workstations. The second version runs using the UNIX operation system and employs either Dextranase or X-Windows workstations. TRADOC Analysis Command at White Sands Missile Range

(TRAC-WSMR) is the Army agency responsible for maintenance and modification of Janus.

This research effort utilized the UNIX based Janus version 4.0 (Janus). This version utilizes polygonal terrain representation which is a significant difference from previous versions of Janus.

1. RAND MODIFICATIONS TO JANUS

The Arroyo Center of RAND is the U.S. Army's federally funded research and development center for analysis and studies. Through the Arroyo Center, RAND was tasked to make modifications to Janus to support its integration with BDS-D.

a. Janus Algorithms

RAND incorporated higher resolution algorithms for target acquisition, Probability of Hit (PH), and Probability of Kill (PK) into Janus. RAND replaced the NVL-1D sensor model with the NVL-2D sensor model and the PH and PK algorithms with those of Army Materiel Systems Analysis Activity's (AMSAA) Groundwars.[JOHNS]

b. Janus Terrain

RAND modified Janus terrain representation to minimize differences between BDS-D and Janus terrain. If terrain representations are different among heterogeneous simulators participating in the same exercise, it is possible for "...game players to 'cheat' by using non-realistic 'blind spots'..."created by inconstancies in the terrain models. [ZOBRIS] The result of the modifications reduced these blind spots, providing a level battlefield among heterogeneous simulators.

c. Adding Hooks to Communicate with the World Modeler

A modular program, called the Name Translator, was built to add communication 'hooks' into the main databases of Janus. This translator adds BDS-D entities into the proper Janus databases. It also retrieves all pertinent Janus entity state and event information and passes it to the World Modeler. [MARTI]

2. HYBRID JANUS 4.0

The result of RAND's modifications is a hybrid Janus combat model that runs on UNIX based, Hewlett Packard Workstations. This version of Janus is used to connect to BDS-D via the World Modeler. All future references to Janus refer to this hybrid Janus 4.0.

B. BATTLEFIELD DISTRIBUTED SIMULATION - DEVELOPMENTAL

BDS-D is an Army program designed to equip the Louisiana Maneuver Battle Labs with the latest distributed interactive virtual reality simulations systems. BDS-D will test future weapon systems and technologies, proposed force structure, and tactics. This concept is expected to save millions of dollars in development costs, and is expected to reduce the time it takes to development systems by a factor of five. BDS-D provides a virtual battlefield where man-in-the-loop simulators and other intelligent platforms can interact and thoroughly test the future technology. If the applied technology appears valid, continued investment occurs. If not, large amounts of money have been saved. [McDon93]

BDS-D represents the most state-of-the-art DIS synthetic environment. Through DIS, geographically dispersed simulations are interconnected and interact in BDS-D. Advance Technology Demonstrations (ATDs) are continually upgrading this environment. Linking Janus to DIS via the World Modeler is one such ATD.

C. NPSNET

Developed and maintained at the Computer Science Department, Naval Postgraduate School, NPSNETIV [PRAT93a] is a low-cost, student written, entity simulator which utilizes the Silicon Graphics IRIS family of computers. NPSNET is compliant with DIS 2.0.3 protocol standards and operates in real time in the BDS-D synthetic environment. [MACED]

NPSNET was chosen as a surrogate crew station for experimentation and testing in the BDS-D environment. NPSNET provides a portal into the BDS-D, allowing us to visually see Janus entity actions in three dimensions. With NPSNET, we are able to view

Janus entities interact with NPSNET and ModSAF entities in BDS-D. The NPSNET simulator proves to be invaluable in the integration of Janus and BDS-D.

D. PREVIOUS WORK

1. Janus-3D

Janus-3D was the first successful attempt of integrating the Janus Combat Modeler with a three dimensional display. This project created a scripting tool which gathered post processing information from Janus. This script was then replayed in three dimensions using NPSNET as the display environment. User's could travel through the Janus battle and see what had occurred. Interaction was not possible. However, this gave Janus analysts the first three dimensional view of a Janus scenario.

This project also made the first attempt of displaying Janus 2.0 entities in real-time in NPSNET. The first non-VAX version had just been produced by TRAC-WSMR in 1992. This first generation UNIX based system (Janus(A)) ran on a Sun Workstation. Janus(A) used ethernet connections to display the two dimensional Janus screens on other monitors. Thus, Janus entity data existed on the ethernet. The next logical step was to capture the data and transfer it into a structured packet that NPSNET understood. The result was a three dimensional, real-time display of a Janus scenario. The user could not interact with the Janus entities. Also Janus did not receive any information about NPSNET entities. [WALT92]

2. Eagle/BDS-D

Eagle/BDS-D is an effort to link the constructive combat model Eagle with the BDS-D environment. This effort follows the successful effort of integrating Eagle with SIMNET, an entity level, three dimensional combat simulator that operates in real-time. Eagle is an aggregate corps/division combat model developed by TRADOC Analysis Command (TRAC). It runs faster than real time and is used as a combat analysis tool. Eagle

units were disaggregated and put into the SIMNET environment. Disaggregated units in SIMNET operated in real-time and battled against other Eagle entities.

The current effort is to link Eagle with the entity level real-time virtual world of BDS-D. Though final testing has not occurred, interim demonstrations of the project have been successful. [KARR].

E. SUMMARY

Previous work has shown that it is possible to integrate heterogeneous combat models. We are going to extend this work to Janus and BDS-D; two disparate Army combat models used to facilitate training and combat development. NPSNET is the DIS-compliant simulator that provides a portal into BDS-D. These three systems form the foundation for constructing the World Modeler.

III. WORLD MODELER OVERVIEW

The World Modeler was designed at the Department of Computer Science, Naval Postgraduate School, as a proof of principle to allow Janus, a monolithic combat model, to interact in the Battlefield Distributed Simulation - Developmental environment (BDS-D). The World Modeler runs on its own Silicon Graphics platform with a TCP/IP connection to Janus and an UDP/IP connection to BDS-D. Janus and the World Modeler communicate with a local protocol; BDS-D and the World Modeler use Distributed Interactive Simulation (DIS) protocols. The architecture chosen for the World Modeler allows for reading and writing to associated message buffers in parallel with the main application loop of the World Modeler. Storage of information is patterned directly after NPSNET, and several NPSNET algorithms are used. In this section, we discuss the software structure used to track information about the virtual world, how we manage network traffic, and how we efficiently control the interaction between Janus and BDS-D entities.

Construction of the World Modeler began in the fall of 1993 as part of the Army's Anti-Armor Advanced Technology Demonstration research effort [A2ATD]. As part of this project, the primary purpose of the World Modeler is to allow interaction between Janus and BDS-D entities. To do this effectively, the World Modeler must perform four major functions: network management, dead reckon Janus entities, reconcile Janus entity locations with the terrain, and arbitrate engagements between Janus and BDS-D entities. Chapters IV through VII discuss these functions in detail. By performing these functions, the World Modeler is the nexus which allows Janus to interact in the BDS-D environment. Figure 1 and Figure 2 show the conceptual and physical architecture of the World Modeler.

The World Modeler runs on a low end Silicon Graphics (SGI) workstation, which allows for maximum efficiency in terms of networking and parallel processing. This choice of hardware allows the use of existing NPSNET software as necessary. Database formats are also consistent with NPSNET. See Appendix A for further hardware and software requirements.

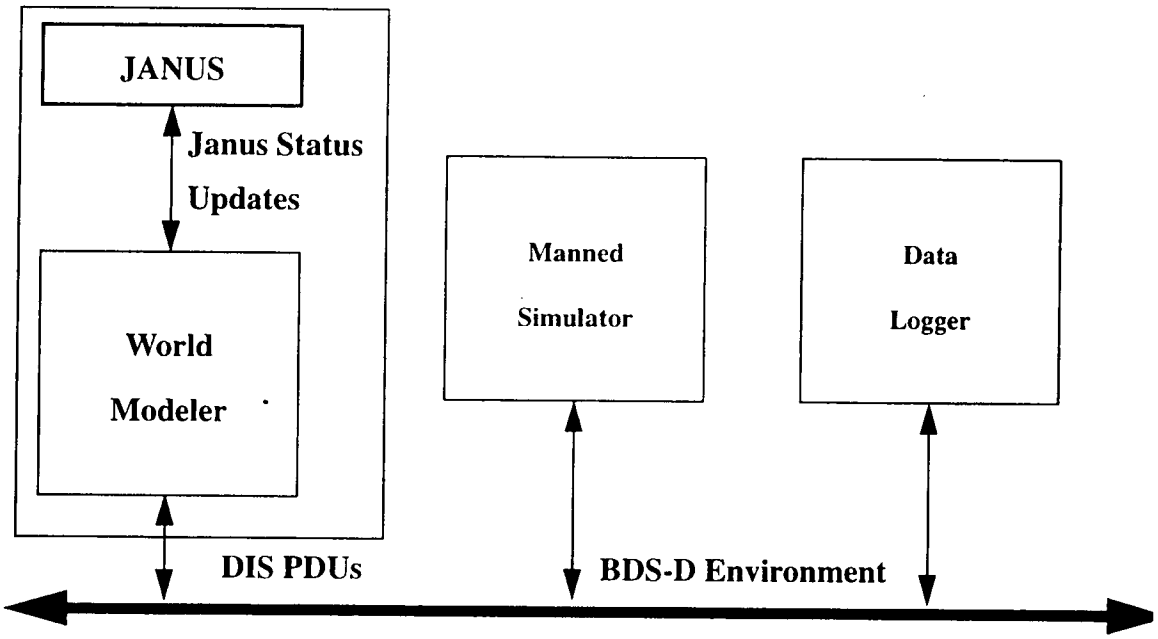


Figure 1. Conceptual Architecture

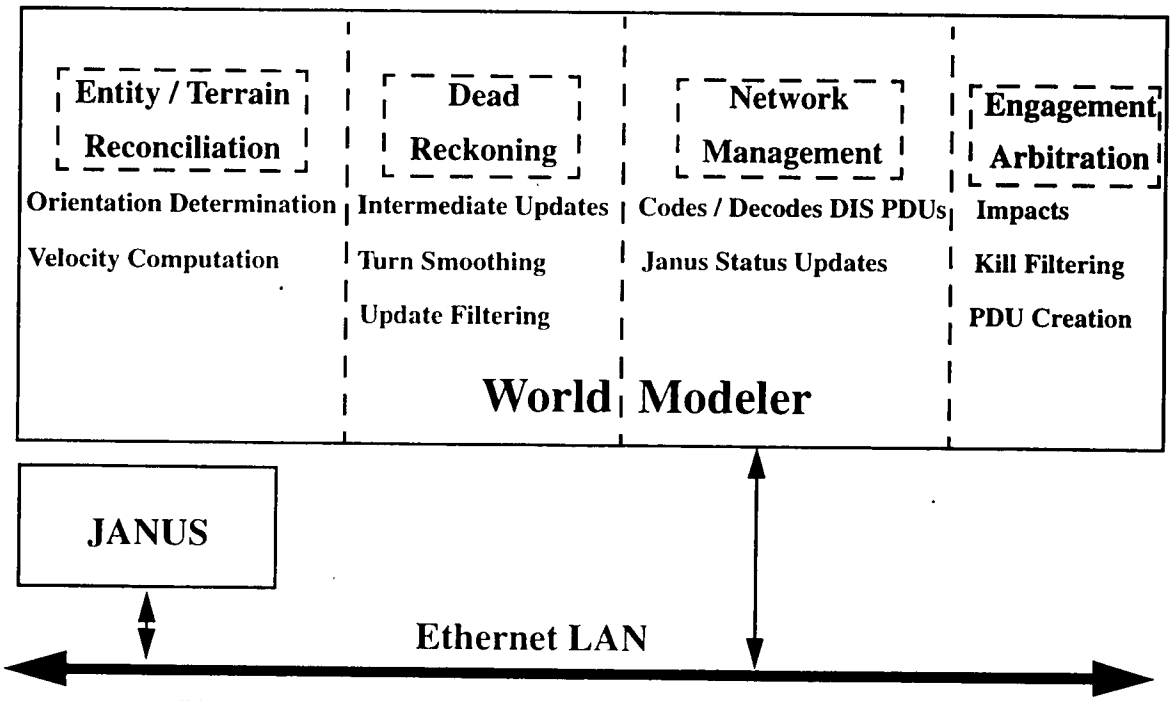


Figure 2. Physical Architecture and World Modeler Functions

A. INITIALIZATION OF THE WORLD MODELER

Before Janus can interact in the BDS-D virtual world, the two different environments of Janus and BDS-D must be reconciled within the World Modeler. The initialization process of the World Modeler performs this reconciliation (Figure 3).

1. Load DIS-JANUS equivalence table into the World Modeler. This datafile contains information on the different entity types that Janus recognizes as part of its simulations. DIS entity type equivalences are matched up to each Janus weapon system. The intent is to ensure that Janus entities are correctly portrayed in the BDS-D environment and that BDS-D entities are correctly portrayed in Janus. For example, we do not want Janus to mistake a BDS-D Soviet T-72 tank with a United States Bradley Fighting Vehicle.
2. Load terrain datafile. Currently, both Janus terrain and BDS-D terrain are derived from the same S1000 terrain database. Janus however, does not explicitly track information on entity elevation values. The World Modeler maintains a copy of the terrain to determine elevation and orientation values for Janus entity positions and munition effects.
3. Establish a TCP/IP connection to Janus and a UDP/IP connection to the BDS-D. Chapter IV discusses each of these connections in detail. Once these connections are established, the associated message buffers begin to fill with Janus and BDS-D entity information.
4. Read the Janus message buffer and load Janus entities into the World Modeler entity array. The array stores information on all Janus and BDS-D entities. Janus entities are flagged to identify them as Janus because all entity information is uniformly stored. These Janus entities are all that will participate in the simulation. Janus does not introduce new entities during the simulation.
5. Read the BDS-D message buffer and load the entity array with the entities currently active in the BDS-D world. Creation and deletion of entities in the BDS-D world is dynamic. The World Modeler only tracks entities that are currently being broadcasted on the ethernet.
6. Send Janus the current BDS-D entity dispositions. Janus acknowledges the

receipt of this data with a synchronization message. This message signifies the completion of the World Modeler initialization. The World Modeler clock starts and the main application loop begins.

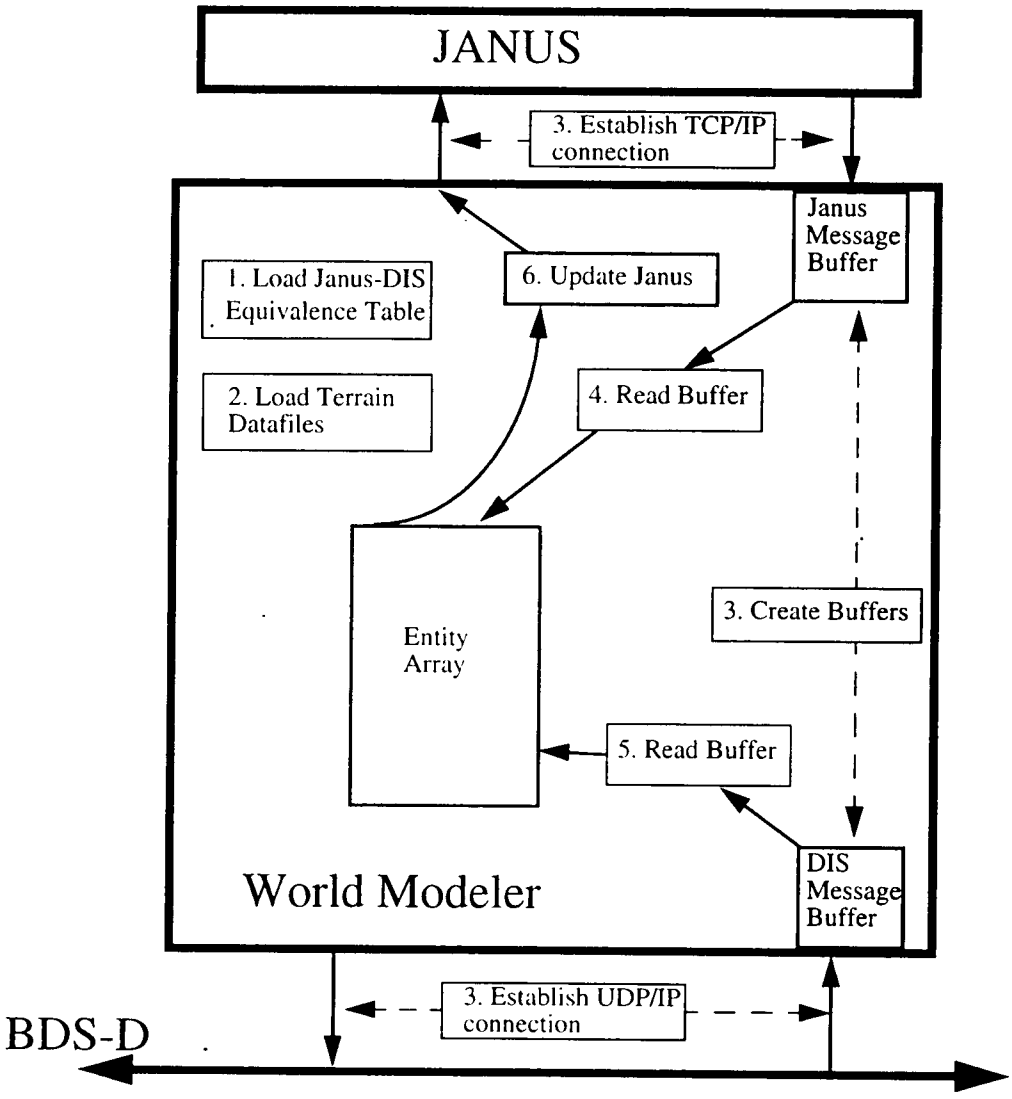


Figure 3. Initialization of the World Modeler

B. WORLD MODELER MAIN APPLICATION LOOP

The main application loop continuously executes the primary functions of the World Modeler (Figure 4). This loop continues until the simulation is terminated either

from Janus or the World Modeler. Another BDS-D application can not terminate the Janus-BDS-D connection.

When the simulation starts, the Janus clock runs a *heartbeat* faster than the World Modeler. The purpose of the heartbeat is described in the following paragraphs. The heartbeat, a variable named JANUS_TIME_INTERVAL, can be set to different lengths prior to the simulation. The heartbeat can not be adjusted dynamically during the simulation. Four seconds appears to be a reasonable heartbeat for scenarios up to 200 Janus entities.

1. Process Janus Messages

The World Modeler reads Janus messages from the Janus message buffer. If the message deals with entity states, then the entity array is updated. If the messages deal with munition effects such as fire and detonations, appropriate Fire and Detonation packet data units (PDU's) are created. Since Janus is running a heartbeat ahead of the World Modeler, these events may not have occurred yet in World Modeler time. Time stamps on the message and the World Modeler clock are compared. If the event occurred, these PDU's are written directly to the Ethernet and broadcasted to the BDS-D environment. If the events have not occurred yet, they are placed in an event queue. The queue is read later in the application loop.

2. Update Janus about BDS-D Entities

The World Modeler keeps track of when it last updated Janus. If the last update occurred in less than a heart beat, Janus is not updated. If the time interval is greater than the heartbeat, the World Modeler updates Janus with information on BDS-D entities. The most current information about BDS-D entity disposition is passed to Janus. Also, any new BDS-D entities that may have entered into the simulation are also passed onto Janus. In this case, the heartbeat is used to limit the number of updates Janus receives. Janus needs time to do internal algorithms such as line of sight and Ph/Pk calculations. Continual updating

of Janus about BDS-D entities may slow Janus down. The heartbeat interval is designed to keep Janus informed but not bogged down.

3. Process BDS-D Entity Information

The World Modeler reads the BDS-D message buffer containing PDU's, parsing the PDU's by type. Entity State PDU's are used to update the entity array. If a new entity has entered the simulation, it is added to the array. Fire and Detonation PDU's are also processed. These PDU's are filtered to see if they have any effect on Janus entities. If they do not, they are discarded. If they effect Janus entities, that information is processed and Janus is immediately updated. This process is discussed further in Chapter VII.

4. Update World Modeler Clock

The current time is assigned to the World Modeler clock. This creates a time delta between the current time and the time that the entity array was last updated.

5. Dead Reckon Entities

The World Modeler dead reckons entities based on their last known position, orientation and velocity. Entity and terrain reconciliation also occurs during this phase. Further discussion of dead reckoning and entity/terrain reconciliation can be found in Chapters V and VI respectively.

6. Update BDS-D Virtual World

The World Modeler is responsible for sending Entity State PDU's to BDS-D within the DIS 2.0.3 standards [IST93a]. To accomplish this task, the World Modeler scans the entity array to determine whether any Janus entities meet the criteria for the generation of an Entity State PDU. If so, the World Modeler creates and sends the PDU. The World Modeler then reads the Janus event queue which contains Janus fire and detonations events in PDU format. If the current time is greater than the event time, the PDU's are broadcasted onto the net. If not, they stay in the queue to be screened the next time through the main application loop.

7. Refresh World Modeler Map

The final portion of the loop is to redraw the World Modeler two dimensional display based on the most current information in the entity array. The cycle then begins again.

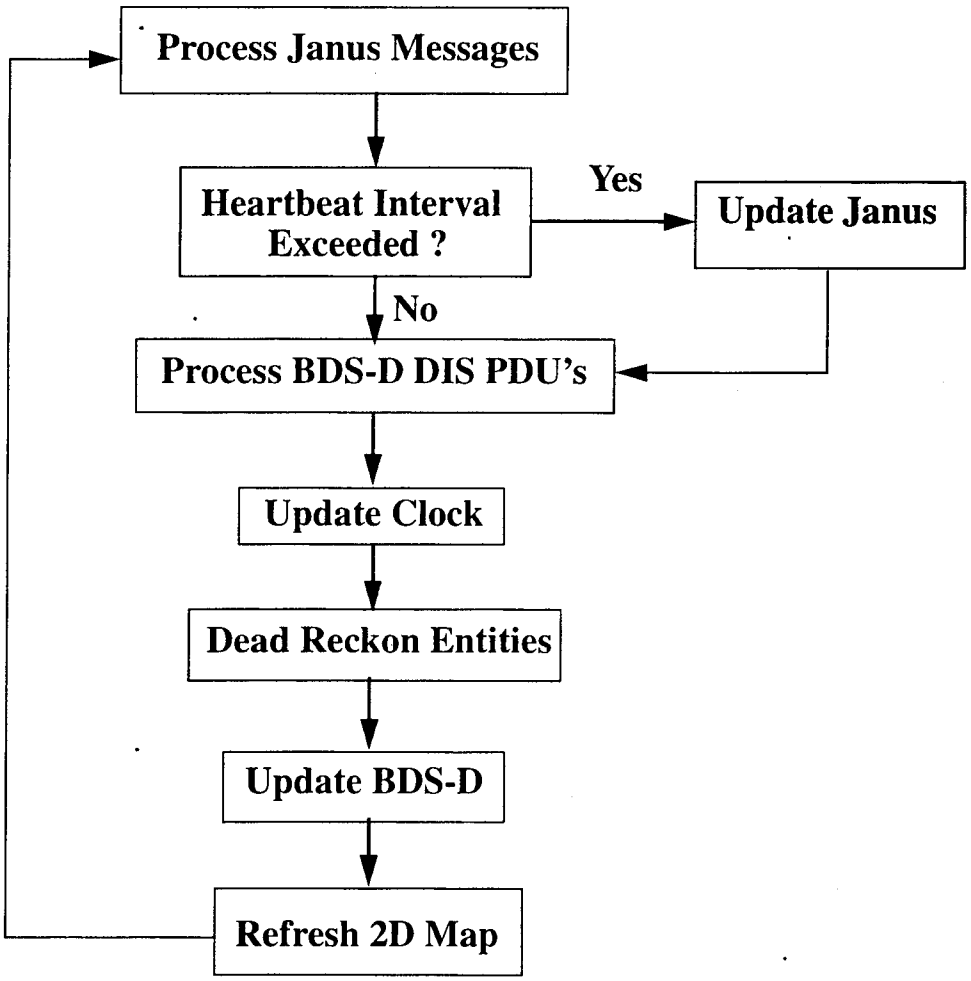


Figure 4. World Modeler Main Application Loop

C. SUMMARY

The World Modeler establishes a ground truth environment to process both Janus and BDS-D entity data and events. Through its process, the World Modeler delivers the necessary functionality to integrate Janus and BDS-D.

IV. NETWORK MANAGEMENT

The World Modeler interconnects Janus and the BDS-D environment, communicating with both simulations in an efficient and timely manner. Without proper communication links, the World Modeler could not complete its other functions. Between Janus and the World Modeler, the Transmission Control Protocol / Internet Protocol (TCP/IP) is used. The World Modeler connects to BDS-D via User Datagram Protocol (UDP/IP).

A. NETWORK ARCHICTECTURE

Both communication connections are established during the initialization of the World Modeler. Associated message buffers are created to store incoming data from Janus and BDS-D. The actual reader processes which load the buffers run in parallel to avoid blocking the main application while messages are coming in. As necessary, the World Modeler reads from these buffers. The World Modeler will write to both network ports as necessary (Figure 5).

B. CONNECTION BETWEEN JANUS AND THE WORLD MODELER

Once established, the World Modeler utilizes the TCP/IP connection throughout the entire simulation. The intent of this communication link is to provide a point-to-point, reliable connection that ensures the delivery and receipt of data between the two systems (Figure 6). TCP/IP ensures retransmission of lost messages through low-level acknowledgment between the two communication hosts [Locke94].

The specific TCP/IP port, a value that distinguishes between networked applications on the same host, must be agreed upon between the machines running the World Modeler and Janus. Currently, the code written defaults to the same port. A command line option exists in the World Modeler to allow the user to set the port number. In Janus, a script file may be modified to change the IP address and port number of the machine running the World Modeler. This allows multiple Janus / World Modeler pairs.

C. CONNECTION BETWEEN BDS-D AND THE WORLD MODELER

Once established, the World Modeler utilizes the UDP/IP connection throughout the entire simulation. The DIS protocol standard requires DIS-compliant combat simulations provide five basic services: data transfer, delivery, best effort service, packet integrity, and minimum performance levels [IST93a]. The NPSNET libraries used, in conjunction with the IRIX network routines, provide the necessary functionality [Zes93]. The current software supports both broadcast (one-to-all) and multicast (one-to-some) communications.

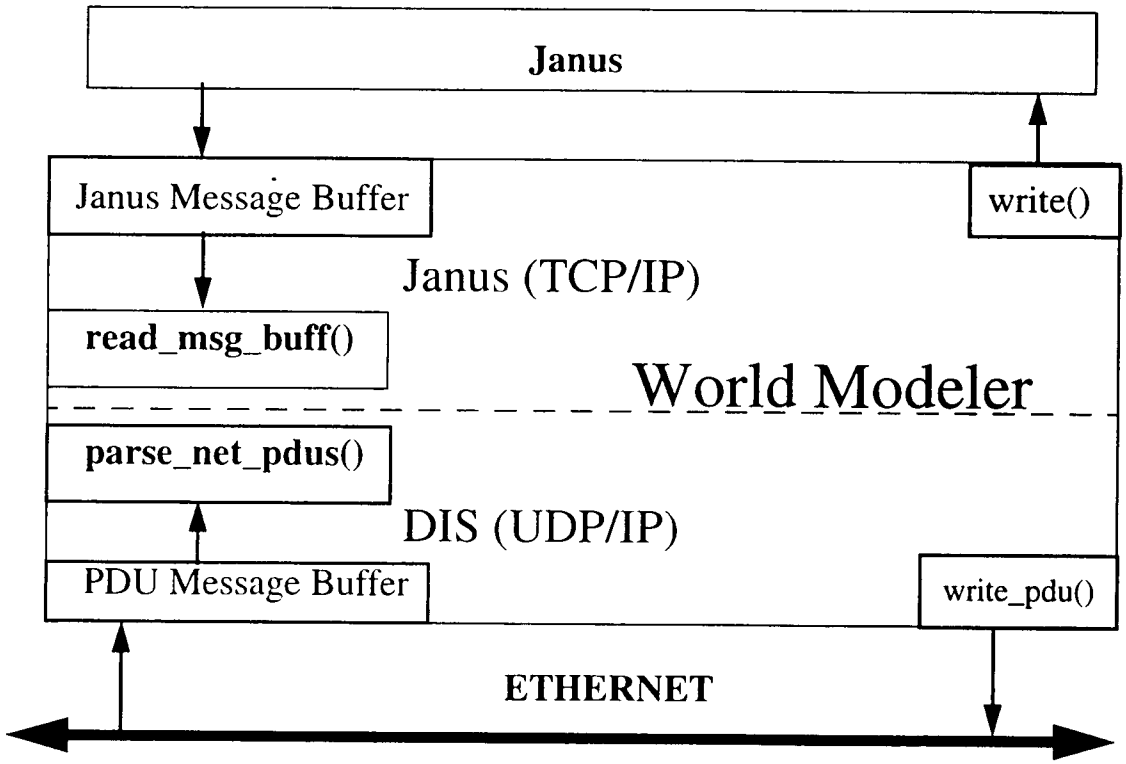


Figure 5. Data Flow in and out of the World Modeler

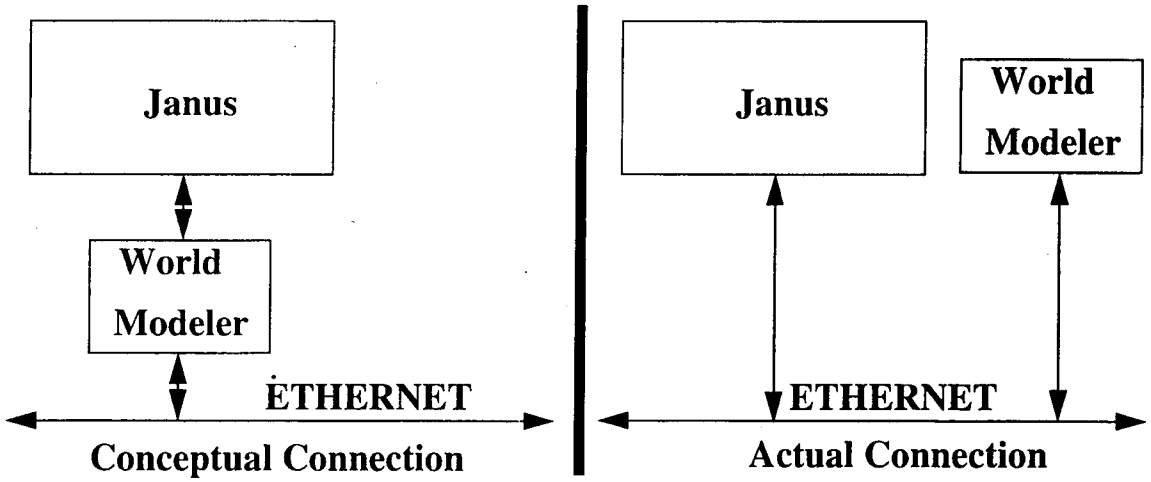


Figure 6. Janus - World Modeler Network Connection using TCP/IP

D. UPDATING THE BDS-D ENVIRONMENT

The World Modeler is responsible for informing the BDS-D environment about all Janus entities and their actions in accordance with the DIS protocol standard. The World Modeler dead reckons Janus entities once every cycle of the main application loop. The World Modeler then updates BDS-D with Janus entity state information if threshold requirements have been exceeded for sending Entity State PDU's. These threshold values are in accordance with Section 7 of [IST93b]. If a threshold is exceeded, an Entity State PDU is sent for that entity. If a Janus entity has not changed state for over five seconds, an Entity State PDU is sent to inform BDS-D that it remains an active entity in the simulation.

Janus also transmits events such as munition firing and detonations. The World Modeler converts these messages to the appropriate PDU type and checks the message time. If the event has occurred in real time, the PDU is written immediately to BDS-D. If it is a future event, the PDU is put on an event queue. The World Modeler reads the queue once every cycle of the main application loop, comparing the current time with the time on the PDU's in the queue. If the current time is greater than the PDU's time, the PDU is written to the net.

E. UPDATING JANUS

Janus does not need updates as often as the BDS-D environment. A heartbeat variable called **JANUS_TIME_INTERVAL** is used to determine whether to update Janus. The intent of the heart beat is discussed in Chapter III, Section B. The World Modeler updates Janus about all DIS entities once every heart beat. IF DIS Fire and Detonation PDU's affect a Janus entity, they are sent to Janus immediately. This is discussed further in Chapter VII.

F. SUMMARY

The World Modeler effectively maintains two network connections, each having a different protocol. Through these links, the World Modeler efficiently manages and controls data flow between Janus and the BDS-D environment.

V. DEAD RECKONING IN THE WORLD MODELER

Dead Reckoning is the process of calculating entity position and orientation based on last known entity posture and predetermined dead reckoning algorithms. The intent of dead reckoning is to limit the number of entity state broadcasts on the network, while still maintaining an accurate posture of remotely-controlled entities. The World Modeler attempts to maintain the most accurate posture of both BDS-D entities and Janus entities.

A. DEAD RECKONING BDS-D ENTITIES

Entity State PDUs (ESPDU) update the World Modeler's entity array with the most current BDS-D entity posture. In between receiving these PDU's, the World Modeler dead reckons a BDS-D entity's posture using the Remote Vehicle Approximation model in accordance with Section 7, of [IST93b].

1. Process Entity State PDU

The World Modeler reads the PDU message buffer and updates the entity array data with the ESPDU data. Dead reckoning parameters such as posture, velocity, and acceleration are assigned from the ESPDU. The PDU also contains information about which type of dead reckoning algorithm should be used for approximating the entity's location. These parameters are used to dead reckon the entity's posture in between receiving PDUs. A variable called **lastPDU** is updated with the current clock time identifying when the entity information was received.

2. Update Entity Positions

The World Modeler updates its clock, and dead reckons the BDS-D entities. The entities are dead reckoned based on their associated dead reckoning parameters and posture. First a time difference is calculated by subtracting **lastPDU** from the current time. Next, the entity's posture is updated based on the dead reckoning parameters and the time difference.(Eq 1.1)

posture = posture + dead_reconking_parameters*delta_time (Eq 1.1)

3. Updating Janus

In between updates from BDS-D, the World Modeler continues to dead reckon the BDS-D entities. The World Modeler updates Janus with the dead reckoned postures.

B. DEAD RECKONING JANUS ENTITIES

The World Modeler is different than other simulators in BDS-D: It does not maintain a high fidelity model of the entities it manages. Janus maintains the high resolution model of its entities, forcing the World Modeler to maintain a 'private' model of Janus entity movement. The World Modeler keeps two entity postures, the private model's posture and the dead reckoned posture using the DIS algorithms. When these values differ by an amount greater than the established DIS dead reckoning thresholds, ESPDUs are sent to update BDS-D with the private model's values.

1. Private Model of Janus Entities

The private model is intended to be a better approximation of Janus entity movement than the DIS dead reckoning algorithms. This model is based on the information that Janus sends the World Modeler. First order linear approximation is used to model the Janus entity movement. The most recent information passed on a Janus entity is considered a future posture. The private model uses this as a target posture for the linear approximation. The Janus velocity vector does not account for the z vector component. Hence, there is some drift in the private model.

2. Calculating Janus Entity Positions

The World Modeler stores Janus entity information in the same manner as BDS-D entities. To keep the entity array uniform, Janus entity messages are converted into Entity State PDUs (ESPDU) and processed the same as a BDS-D ESPDU. However, the Janus entity information is used differently in order to maintain both the private and dead reckoned models.

Figure 7 shows the process of updating the two models' postures and sending out ESPDUs as necessary. The variable **drpos** contains the dead reckoning model's posture of the entity. **drparam** variables are used to calculate the dead reckoned model's posture. This model represents the model that other BDS-D simulators use to dead reckon Janus entities.

The data member **posture** contains the private model's posture. **target** variables are used to calculate the private model's projected posture. The variable **remaining_time** is the time step used for first order linear smoothing of the private model's velocity vector.

3. Updating BDS-D

During the function call **moveDR()**, the function **delta_send()** determines if the posture values of the private and dead reckoned model exceed the thresholds. If so, an Entity State PDU is sent for that Janus entity.

C. SUMMARY

The World Modeler dead reckons BDS-D entities in accordance with the current DIS standards and Janus is updated with these dead reckoned postures. For Janus entities, the World Modeler maintains a private model and a dead reckoned model. The private model is a first order approximation of the Janus movement. When the private and dead reckon models' postures exceed threshold levels, the World Modeler issues an Entity State PDU for that Janus entity in accordance with DIS specifications.

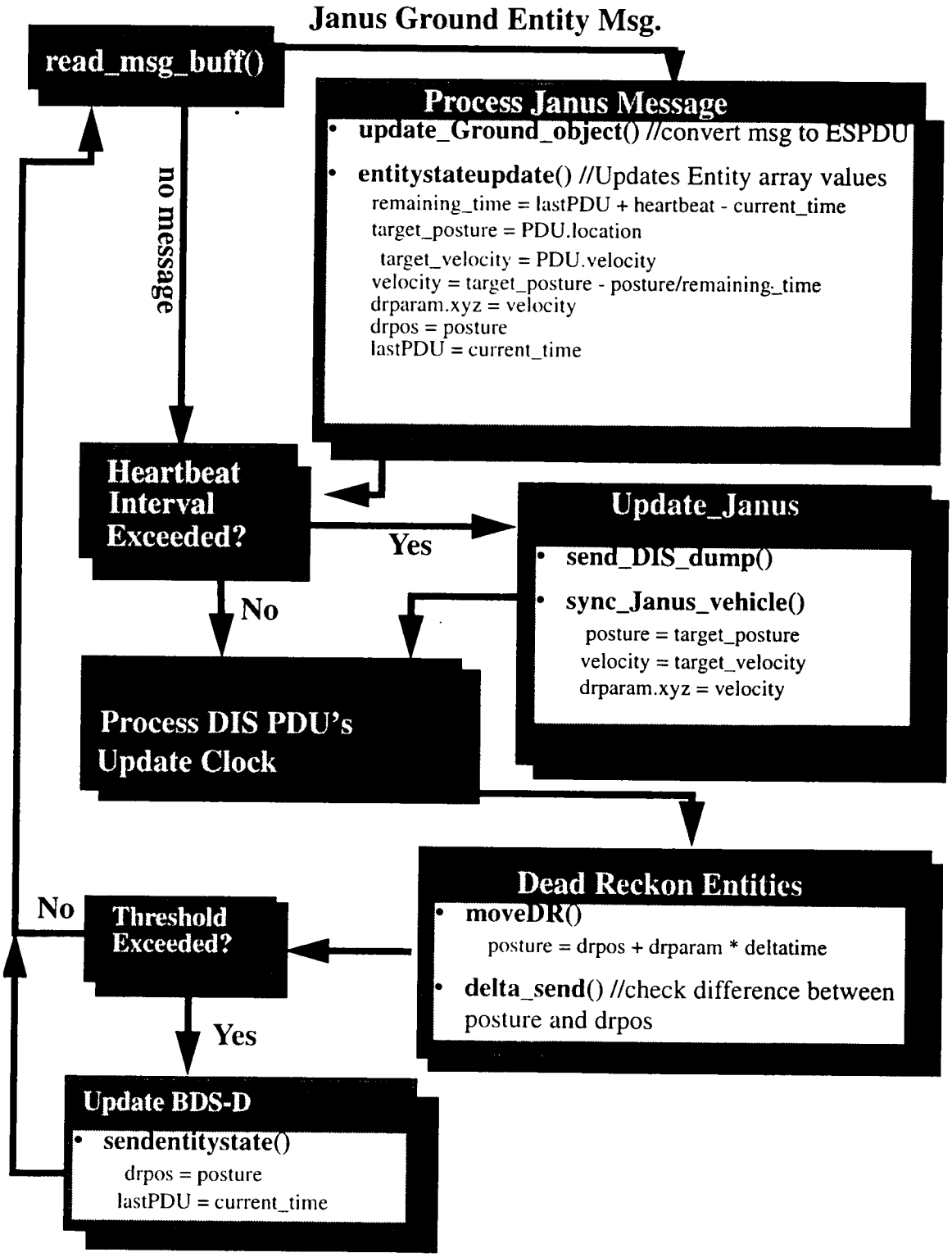


Figure 7. Maintaining the Private and Dead Reckon Postures of Janus Entities

VI. JANUS ENTITY/TERRAIN RECONCILIATION

Entity/terrain reconciliation is the World Modeler function that resolves the differences between the BDS-D visual terrain databases and the Janus terrain files. There are three fundamental tasks to this function. First is the computation of entity orientation in the BDS-D environment. Second, to ensure compliance with the DIS standard, entity speed is converted to a three dimensional velocity vector. Third, Janus UTM coordinates are transformed to BDS-D geocentric Cartesian coordinate system.

A. COMPUTATION OF JANUS ENTITY ORIENTATION

Janus does not track entity pitch and roll. To give realism to the appearance of entities in manned simulators of BDS-D, a copy of the visual database is maintained in the World Modeler. Entities are "snapped" to the visual terrain, i.e., the elevation coordinate, pitch and roll of the underlying terrain is assigned as the entity's orientation. This ensures visual consistency with the manned simulator.

1. Janus Entity Posture

Janus maintains easting and northing position values using the UTM coordinate system with the distance of one unit equal to a kilometer. Janus maintains only entity headings in radians with zero radians to the East. Positive rotation is in the counter-clock direction.[JANU93] Elevation, pitch, and roll values are not explicitly kept.

2. BDS-D Entity Posture

BDS-D, in accordance with DIS Standards, maintains x, y, and z values using the right-handed Geocentric Cartesian coordinate system, referred to as the World Coordinate System. One unit is equal to a meter. An entity's orientation in BDS-D is defined by three Euler angles in radians, using the right-handed coordinate system, and with positive rotation in the clockwise direction. [IST93a]

3. World Modeler Entity Posture

The World Modeler must reconcile these two disparate methods of describing an entity's posture. To do this, the World Modeler maintains all entities, Janus and BDS-D, in the same fashion. Position values, referred to x, y, and z, are stored using the UTM like coordinate system with one unit equal to a meter. Entity orientation is defined by three Euler angles in degrees, using the right-handed coordinate system, and with positive rotation in the counterclockwise direction. This method of describing an entity's posture was chosen to take advantage of existing NPSNET algorithms.

4. Conversion Between BDS-D and World Modeler Posture

When a BDS-D Entity State PDU is read from the message buffer, the network code has already converted the location to UTM coordinates in meters (see Section C of this chapter). The World Modeler still has to convert the Euler angles. The following equations are used to make the conversions. **RAD_TO_DEG** is an eight decimal precision float value to convert radians to degrees.

```
posture.hpr[HEADING] = (epdu->entity_orientation.psi > 0)?
        - (epdu->entity_orientation.psi + 2*M_PI)*RAD_TO_DEG:
        - (epdu->entity_orientation.psi)*RAD_TO_DEG;
posture.hpr[PITCH]   = epdu->entity_orientation.theta*RAD_TO_DEG;
posture.hpr[ROLL]    = epdu->entity_orientation.phi*RAD_TO_DEG;
```

The reverse process is done when a Janus entity's orientation is sent out as part of a PDU. **DEG_TO_RAD** is an eight decimal precision float value to convert degrees to radians.

```
epdu.entity_orientation.psi   = -posture.hpr[HEADING]*DEG_TO_RAD;
epdu.entity_orientation.theta = posture.hpr[PITCH]*DEG_TO_RAD;
epdu.entity_orientation.phi   = posture.hpr[ROLL]*DEG_TO_RAD;
```

5. Conversion Between Janus and World Modeler Posture

When a Janus entity message is read from the message buffer, it only contains the x, y and heading values of the entity's posture. The World Modeler, using the functions in **msg_code.C**, converts these messages into ESPDU's. First, the x and y values are

multiplied by one thousand to convert them to UTM units that are in meters. The heading is converted to degrees and reduced by ninety degrees. This is done to take advantage of the NPSNET function, **grnd_orient()**, which calculates the pitch, roll, and elevation values based on the x, y and heading values. The elevation value for a ground entity is given the elevation value of the terrain at the given x and y location. The proper pitch and roll values in degrees is returned. The orientation components are converted back to radians; the heading is negated. Now the posture is described in the same fashion as a BDS-D entity. This Janus Entity State PDU is then processed in the same fashion as a BDS-D ESPDU.

The reverse process is used to convert BDS-D entity information, which is stored in the World Modeler fashion, to Janus posture fashion. Only the x, y and heading values are passed since this is all that Janus uses.

B. CREATING A THREE DIMENSIONAL VELOCITY VECTOR

BDS-D ESPDUs arrive with all three component velocity vectors in meters per second. The World Modeler maintains the information the same way and direct assignment from the PDU occurs.

Janus messages arrive with a velocity vector which is in kilometers per hour. Functions in **msg_code.C** convert the two dimensional velocity vector into its x and y components in units of meters per second. Zero is assigned to the z velocity vector. This information is assigned the Janus Entity State PDU which is processed in the same fashion as a BDS-D ESPDU.

When it is time to update Janus about BDS-D entities, the World Modeler computes the magnitude of the velocity vector. This value is passed to Janus.

C. COORDINATE CONVERSION

Janus and BDS-D terrain are derived from the same S1000 terrain database. The World Modeler maintains a copy of the terrain to determine elevation and orientation values for Janus entity positions and munition effects.

1. Grid Coordinate Translation Between Janus and the World Modeler

The origin of the Janus coordinate system is located at the lower-left corner of the map, labeled with UTM coordinates. The terrain database that the World Modeler uses is also in the lower-corner, but in a different UTM coordinate. To resolve this difference, the World Modeler's origin is translated to the Janus origin, in meters. The following equations show how the World Modeler origin is translated to equal the Janus terrain origin.

```
G_WM_origin.easting = G_WM_origin.ter_east - G_WM_origin.jan_east;
G_WM_origin.northing = G_WM_origin.ter_north - G_WM_origin.jan_north;
```

2. Converting Entity Positions Between Janus and the World Modeler

When a Janus message is read by the World Modeler, the easting and northing values of the entity are multiplied by 1000, converting their units to meters. The World Modeler origin is then added.

```
position.x = (mess->easting)*1000 + G_WM_origin.easting;
position.y = (mess->northing)*1000 + G_WM_origin.northing;
```

When a message is written to Janus from the World Modeler, the opposite conversion occurs.

```
mess.easting = (position.x - G_WM_origin.easting)/1000.0f;
mess.northing = (position.y - G_WM_origin.northing)/1000.0f;
```

3. Converting Entity Positions Between World Modeler and BDS-D

Network level code converts incoming geocentric Cartesian coordinates to UTM coordinates as they arrive. When the World Modeler reads the PDU message buffer, all positions are in the correct format. When the World Modeler sends PDUs to BDS-D, the network code converts the out going coordinates from UTM to geocentric coordinates.

D. SUMMARY

The World Modeler successfully resolves the disparity between the manner that Janus and BDS-D describe an entity's posture. The World Modeler also supplies the

missing elevation, pitch and roll values to the Janus entities. The result is Janus entities appearing properly in BDS-D.

VII. EVENT ARBITRATION

Event arbitration is the function of the World Modeler that resolves the relationship between Janus and BDS-D entities when they are shooting at each other. Four different possibilities can arise which the World Modeler must arbitrate: Janus targeting a Janus entity, Janus targeting a BDS-D entity, BDS-D targeting a Janus entity, and BDS-D targeting a BDS-D entity.

One of the integral aspects of DIS architecture is that a targeted entity determines the effect of the munition's impact. This differs from Janus, which is a self-contained combat model, arbitrating interaction between all of its entities. The implementation requires Janus to track BDS-D entities. However, Janus is not able to move or destroy them. Janus must work through the World Modeler to interact with the BDS-D entities. Thus, BDS-D entities tracked by Janus are referred to as 'ghosts.' The following engagements refer to both direct and indirect fire engagements.

A. JANUS SHOOTS BDS-D

Janus sends the World Modeler a fire and detonation message when it shoots a BDS-D entity. The World Modeler creates Fire and Detonation PDU's and broadcasts them to the BDS-D world. The simulator controlling the targeted entity determines the effect of the munition's impact. This effect is reflected in the subsequent Entity State PDUs (ESPDU) from the targeted vehicle. If the vehicle was destroyed, the World Modeler updates Janus, which in turn updates the representation of the ghost vehicle within Janus (Figure 8).

B. JANUS SHOOTS JANUS

In this situation, Janus does determine the results of the engagement. The fire and detonation message still gets passed to the World Modeler, which creates the appropriate PDU's. This allows the visual effects of the engagement to be displayed in BDS-D. If the targeted vehicle is destroyed, subsequent ESPDUs reflect the change in entity state.

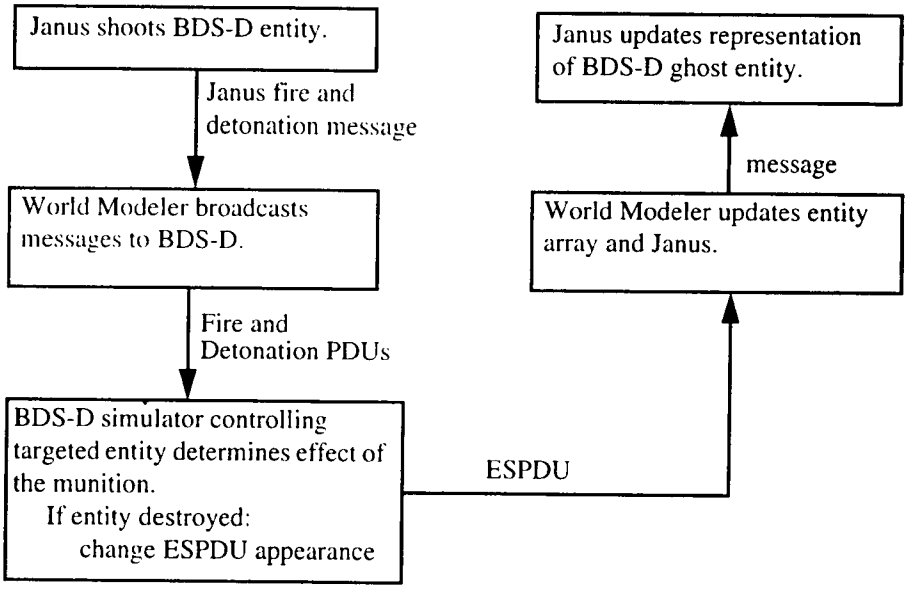


Figure 8. Janus Shoots BDS-D Entity

C. BDS-D SHOOTS JANUS

BDS-D simulators generate a Fire and Detonation PDU when it shoots a Janus entity. Current implementation has the World Modeler determining the effects of the impact. The World Modeler tells Janus that the entity is dead. Subsequent ESPDUs of that Janus entity reflect the change in status. Future implementation will have Janus receive the fire and detonation information to make its own determination of the munition's effects.

D. BDS-D SHOOTS BDS-D

When BDS-D entities engage each other, the World Modeler filters out the Fire and Detonation PDUs. Hence, Janus does not receive any information on the engagement. Janus receives the updated entity state information on the targeted vehicle based on the engagement's outcome. If the entity is destroyed, Janus changes the representation of the internal ghost.

E. JANUS MUNITION SELECTION

Janus and the World Modeler both have the same equivalence table which stores Janus munition types and their DIS equivalents. When Janus fires a weapon, the munition is cross-referenced in this table so that the corresponding DIS munition entity type is broadcasted to the net.

If Janus fires a precision guided munition, the World Modeler generates ESPDUs to visually depict the missile flying in BDS-D. At impact, the missile is destroyed by the World Modeler.

F. SUMMARY

The World Modeler successfully generates the necessary PDUs to allow Janus to interact with BDS-D entities. As per DIS standards, Janus does not kill ghost BDS-D entities within Janus until updated by the World Modeler. The World Modeler currently determines if a BDS-D entity destroyed a Janus entity. Future implementation strives for Janus to determine the fate of its own entities.

VIII. RESULTS

A. PERFORMANCE

There are several aspects of performance that can be measured in reference to the World Modeler. The areas of analysis can vary from the network connections to handling of data within the World Modeler. We have chosen to limit our scope of evaluation to the output of Entity State PDUs (ESPDU) and the visual display of Janus entities in BDS-D.

1. Testing World Modeler Entity State PDU Output

Janus updates the World Modeler infrequently. This makes the World Modeler responsible for sending ESPDUs to BDS-D in accordance with DIS standards. The World Modeler was tested to ensure that the DIS standards were met.

a. Methodology

NPSNET was used to visually see the Janus entities in BDS-D. The World Modeler ran on a Silicon Graphics Indy workstation with a 100 MHZ IP22 Processor and 64 Mbytes of main memory. The Janus update interval was set to four seconds.

Different scenarios were used to test ESPDU output. All scenarios were run for five minutes (Table 1).

Scenario	Red Ground	Red Air	Red Soldier	Blue Ground	Blue Air	Blue Soldier	Total	Interaction
A	1	0	0	1	0	0	2	None
B	0	1	0	6	3	5	15	None
C	19	0	0	17	0	0	26	Combat
D	96	17	10	40	7	28	198	Combat

TABLE 1: Scenarios Used to Test Entity State PDU Output

Scenarios A and B were used to test ESPDU output for static and moving Janus entities; with and without turrets rotating. Scenarios C and D are more in line with an actual Janus scenario. They were used to analyze the scalability of ESPDU output.

The DIS standards state that the local simulation manager determines the thresholds for sending out ESPDUs. When the standard is not set, default values are used. Default values include: one ESPDU every five seconds for static vehicles; one ESPDU every time the entity has moved more than a meter from the last ESPDU position sent; one ESPDU every three degrees of rotational movement. [IST93a]

The default standard of one ESPDU for every meter change in position seemed excessive. For the tests, we chose to increase the position threshold to three meters and made an additional test run at the higher one meter standard. The intent here was to see if the visual output was degraded by this change in threshold.

b. Results

The World Modeler met the default standards for sending out the necessary ESPDUs (Table 2). There was no visual degradation using the three meter position threshold and ESPDU output was decreased by almost a third (compare runs 3 and 4). Comparing runs 3 and 5, it is noted that there was no major increase in ESPDU output for Janus entities that were moving with their turrets rotating. Because of this, the remaining runs include rotating turrets.

Scenario B had some entities that did not have turrets. There was a slight decrease in the total ESPDU output for this larger scenario (Compare runs 2 and 6; and runs 3 and 7). Run 7 has almost double the number of PDUs per second compared to run 3. This can be attributed to the fact that one fourth of the entities in run 7 were air vehicles. These vehicles move quicker which cause ESPDUs to be sent more often.

Runs 9 and 10 were combat scenarios. The ESPDU traffic was less because not all entities are moving in an actual planned scenario; and killed entities do not move. These are more likely output levels to expect, since they resemble actual Janus scenarios.

The main application loop must stay below the Janus update interval. In these tests, the application loop time stayed well below the interval of four seconds.

Run	Scenario position threshold equal to three meters unless otherwise noted	Activity	ESPDU's/ Second	Average ESPDU's per Entity per Second	Application Loop Time in Seconds
1	A	static	0.394	0.197	0.014
2	A	static; rotating turrets	1.280	0.640	0.014
3	A	moving; turrets static	6.066	3.033	0.014
4	A posture threshold equal 1 meter	moving; turrets static	17.700	8.850	0.014
5	A	moving, rotating turrets	17.70	3.080	0.014
6	B	static, rotating turrets	8.145	0.543	0.015
7	B	moving, rotating turrets	88.800	5.920	0.020
9	C	small engagement	80.028	2.223	0.021
10	D	large engagement	161.568	0.816	0.200

TABLE 2: World Modeler Entity State PDU Output

2. Visual Display of Janus Entities in BDS-D

Janus entities were viewed in the BDS-D environment through the portal provided by the NPSNET stealth vehicle. Methodology for testing was by viewing the entity movement and judging if it looked realistic to the man-in-the-loop.

For runs one through nine, the entities all appeared to move in real-time with very minimal jumping. The ground entities appeared to be moving along the ground. When they were moving up and down a hard slope, they did move into the ground. This arises from the dead reckoning problems with the elevation velocity vector. Entities do not make smooth turns. They make abrupt turns at the nodes where Janus changes the entity's heading.

Run ten caused the frame rate of NPSNET to slow down to an average of 0.6 frames per second. This made the Janus entity movent not appear in real-time. We attributed this to the viewing application's ability to process the number of entities in scenario D.

Overall, the movement appeared realistic. A man-in-the-loop probably would have judged the Janus entities to be other man-in-the-loop simulators. Visual discrepancies could have been excused to the portal that viewed the BDS-D environment.

B. ACHIEVEMENT OF GOALS

Janus entity movement appears in real time. The World Modeler is able to manage large number of entities, keeping both Janus and BDS-D updated with the other entities. However, Janus entity response to BDS-D entities appears slow. When an enemy BDS-D entity drives up to a Janus entity, the Janus entity will detect its presence immediately, but not take action for sometimes up to twenty seconds. We attribute this to internal Janus algorithms and something to look at for future work. BDS-D entities can shoot and kill Janus entities in real time.

C. SUMMARY

The major goal of integrating Janus entities in BDS-D was achieved. The World Modeler successfully performs its functions, as outlined in Chapters IV though VII, allowing real-time interaction between Janus and BDS-D entities. This interaction occurs in both the BDS-D environment and in the Janus Combat environment.

IX. CONCLUSIONS AND AREAS FOR FUTURE RESEARCH

A. CONCLUSIONS

This research effort shows that closed combat models such as Janus can be enhanced to interact with DIS-compliant combat simulations through a software interface.The World Modeler is the interface which provides connectivity, data translation, and event arbitration between Janus and BDS-D in real-time.

B. SPECIFIC CONTRIBUTIONS OF THIS WORK

The World Modeler provides insight as to whether a stand alone interface versus a built-in interface is the appropriate approach to providing interoperability between the two disparate combat simulation models. Though not seen in this research effort, processing time within the World Modeler may become an issue when large combat simulations occur.The current implementation provides for distribution of data processing and minimal modifications to Janus.

The Janus combat model is enhanced by this integration of BDS-D. Previously, the only human interaction provided by Janus was the ability to position and maneuver forces through the puck interface. Choice of targets, engagement criteria, and weapons selection was all determined by Janus. The BDS-D integration provides the added value of human initiative into the Janus environment. This paramount combat multiplier now can be studied in the Janus combat model.

Incorporation of Janus into BDS-D allows the introduction of entities controlled by an accredited combat model to interact in the DIS environment.This provides a means of generating large numbers of entities in BDS-D for training and tactical development purposes. Also, this provides a visualization tool for analysts.

C. AREAS OF FUTURE REASEARCH

1. Implement Reactive Behaviors

Janus entities should look and act as realistic as possible in relationship to man-in-the-loop combat simulators operating in BDS-D. Currently, Janus entities do not show reactive behaviors in relationship to their environment. Janus entities go where they were scripted in the Janus scenario. They drive through trees, other entities, and make no reactive behavior to incoming fire. Janus entities could be made to look more realistic in BDS-D if obstacle avoidance behavior and battle-drill type behavior were incorporated in the Janus model. Whether this change is something that should occur in the World Modeler or in the Janus model is an interesting issue.

2. Increased Fidelity of Discrete Event Arbitration

Current implementation of event arbitration does not address the issue of closely coupled Janus and BDS-D entity interaction. Current update intervals of the BDS-D environment to Janus is four seconds. This means that Janus may wait up to four seconds before it is informed that one of its entities is being attacked. A minimum of another four seconds expires before Janus sends a response to the event. The World Modeler clock, running up to four seconds behind Janus, waits to send the message to BDS-D until the Janus message time is equal to the World Modeler clock. Thus, a minimum of twelve seconds elapses between the BDS-D action and the Janus reaction. This makes Janus entities an easier target in BDS-D. Future work could analyze the current architecture for handling event arbitration and propose alternative techniques to enhance closely coupled Janus/BDS-D engagements (Table 3).

Event	Time in seconds
World Modeler receives Fire PDU	0
World Modeler updates Janus	0 - 4

TABLE 3: Janus Response Time to BDS-D Action

Event	Time in seconds
Janus sends response to World Modeler	4 - 8
World Modeler sends response PDUs	4
Total time elapsed	**8 - 16**

TABLE 3: Janus Response Time to BDS-D Action

3. Sensor Modeling

Current message traffic between Janus and BDS-D is limited to weapon's fire, detonation of munitions, and entity state information. Janus and BDS-D provide for a large number of sensor data that could be tapped into. Implementation would be similar to the current technique of processing the other messages. This would enhance and increase the robustness of both combat models.

APPENDIX A. WORLD MODELER USER'S GUIDE

Janus and the World Modeler work together to allow Janus to interact in the BDS-D environment. A set of procedures is used to start both Janus and the World Modeler. Both Janus and the World Modeler have the ability to time out while waiting for the TCP/IP connection between them. Because of this, the following procedures was outlined to insure that someone unfamiliar with the Janus-BDS-D connection could start the processes. These procedures are based on Janus being on a Hewlett Packard Unix based platform. The World Modeler is on an SGI platform with Unix 5.2 and Performer1.2. Recommended hardware for the World Modeler is a SGI Indy R4000 SC and 64MBytes of main memory.

A. STARTING JANUS

1. Login on the Hewlett Packard which runs Janus 4.0 with Rand modifications.
2. A blue c shell window should pop up. If so, continue to step 3. If not:
   - First you need an xterm or c shell. To do this, click on the computer icon located on the bar on the bottom of the screen. This gives you an hpterm.
   - Type 'jwan'. This is an alias that gives you a c shell. You can get a larger font by putting the mouse in the c shell window and pressing the control key and the right mouse button. You do not need the hpterm anymore. Do the rest of the work out of the c shell.
3. You should be in the /work/janus directory. Change directory to the Programs directory. You are now in /work/janus/Programs directory.
4. Edit the file JLINKWM.DAT.
   - Find "network worldmodeler NPSWM <machine name> 3001 1".
   - Change the machine name to the machine you plan to run the World Modeler on. Example: "network worldmodeler NPSWM elsie 3001 1".
   - Save the file.
5. In your c shell, type 'janus' and hit return.
6. Enter 126 for the scenario and hit return. You can run any scenario; 126 is good

for your basic demo. 133 or 134 is also good for your basic demo.

7. Enter 99 for the run and hit return.
8. Press enter to continue.
9. Press the 'Numeric Enter' three times. Janus is now initializing.
10. Click on the two Janus icons and resize the windows.
11. When initialization is complete, the maps will be displayed in both Janus windows and a control panel will appear on the ride side of both screens.
12. You can find the word 'Start' in the lower right portion of the control panel of both Janus screens. Click on the word 'Start' in both screens.
13. Move mouse to the c shell that you started Janus in.
14. Type 'RR' and hit enter.
15. Type 'n' and hit enter.
16. Now Wait! Go and start the World Modeler before you go any further.

B. STARTING THE WORLD MODELER

1. On the SGI machine that you referenced in the JLINKWM.DAT file, change directory to ~janus/world_modeler.
2. Type 'worldm' and hit enter. Note, this will default to exercise id 5. If you want to run on an other exercise, add the exercise number to the command line. Example: if you want exercise 3, type 'worldm 3'.
3. The World Modeler is now initializing. Size the two dimensional window and stand by.
4. When the 2D window changes color and states "Waiting for Connection on port 3001", go to Thor where you are running Janus.
5. Put mouse in the c shell that you started Janus.
6. Press enter and the connection between the two machine is established.

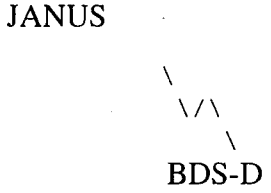
C. BRINGING THE SYSTEMS DOWN.

If everything is still running and you want to bring to quit, go to the World Modeler, put the mouse in the 2D screen, and press the escape key. This should stop both the World Modeler and Janus. If one of the systems crashed, do the following for each machine.

1. For the World Modeler, put the mouse in the 2D window, and press the escape key. If this does nothing, go to the window that you started the World modeler and type 'slay' and enter. This should bring down both the World Modeler and Janus.
2. If Janus did not come down, type ^C in the shell you started Janus. Now type 'slay'. You are ready to start again.

APPENDIX B. PROTOCOL BETWEEN JANUS AND THE WORLD MODELER

This appendix contains the protocol message structure used for communication between Janus and the World Modeler.




```
World Modeler JANUS link message protocol.
Phases:
  **Janus initializes WM** - JANUS initializes the WM with its objects.
  **WM initializes Janus**  - WM initializes JANUS with DIS objects.
  **Janus Updates WM**   - JANUS status to the world modeler.
  **WM updates Janus**    - WM synchronizing JANUS.
/* note: Janus will provide the basic count of entities (generic_index)
after that the world modeler can start counting at (generic_index +1) */
#define INVALIDOBJECT -1
/* Until we know better, this is the DIS maximum name length + 1 */
#define DISNAMELENGTH 16
/* Define a rather large input buffer and do our own fiddling. */
#define BUFFERMAX 8192
/* This is the maximum number of nodes (i.e. NUMNODES, see globnode.for) */
#define NUMNODES 50
/* Define this value to generate logging data. This is not yet complete
 everywhere but should be added soon. */
#define LOGGING 1
/* This is the maximum number of submunitions for indirect fire. */
#define MAXSUBMUNITIONS 20
/* SYNC Message - used for various purposes to tell different processes that
 we've completed whatever it is we're doing. We have different sync
 message types for various purposes. */
struct msg_sync {
 int    msg_len;       /* Length of this msg (excluding msglen) */
 int    msg_type;      /* Msg_Sync */
};
```

```
/*****************************************************************************
                    JANUS <-> WM INITIALIZATION

  Initialization of JANUS - BDS-D objects. These messages are sent during the initializa-
tion sequence only.

*****************************************************************************
*/

/* Direction: JANUS -> WM
   Purpose: To establish that JANUS and the WM are on the same playing
    field. I've just put some random stuff in here, perhaps there should be others.
   When: **Janus initializes WM**
*/
#define Msg_JANUSWM_Handshake 1000
struct msg_januswm_handshake {
  int    msg_len;       /* Length of this message (excluding msglen) */
  int    msg_type;      /* Msg_GroundObject */
  int    easting_offset;  /* Easting offset in meters. */
  int    northing_offset; /* Northing offset in meters. */
};

/* GROUND OBJECT INITIALIZATION
   Direction: JANUS -> WM
   Purpose: Initialize a ground object instance in the world modeler and
    DIS. A ground object is one that doesn't fly, swim, or blow up.
   When: **Janus initializes WM**
*/
#define Msg_GroundObject 1020
struct msg_groundobject {
  int    msg_len;      /* Length of this message (excluding msglen) */
  int    msg_type;     /* Msg_GroundObject */
  int    type_index;   /* Common type index (from data file). */
  int    generic_index;  /* The generic object index. */
  float  easting;      /* Kilometers easting from southwest corner. */
  float  northing;     /* Kilometers northing from southwest corner. */
  float  heading;      /* Course radians (0 is east) */
  float  velocity;     /* Velocity in KPH */
  short  status;       /* Status (see below). */
  short  flags;        /* Object flags */
};
```

```
/* AIR OBJECT INTIALIZATION
   Direction: JANUS -> WM
   Purpose: Initialize an air object instance in the world modeler and
    DIS. An air object flys and has those attributes.
   When: **Janus initializes WM**
*/
#define Msg_AirObject 1021
struct msg_airobject {
  int    msg_len;       /* Length of this message (excluding msglen) */
  int    msg_type;      /* Msg_GroundObject */
  int    type_index;    /* Common type index (from data file). */
  int    generic_index; /* The generic object index. */
  float  easting;       /* Kilometers easting from southwest corner. */
  float  northing;      /* Kilometers northing from southwest corner. */
  float  heading;       /* Course radians (0 is east) */
  float  velocity;      /* Velocity in KPH */
  float  agl;           /* Above ground level. */
  float  roll;          /* Roll of the Object*/
  float  pitch;         /* Pitch of the Object*/
  float  yaw;           /* Yaw of the Object*/
  short  status;        /* Status (see below). */
  short  flags;         /* Object Flags */
};

/* WATER OBJECT INITIALIZATION
   Direction: JANUS -> WM
   Purpose: Initialize a water object instance in the world modeler and
    dis.
   When: **Janus initializes WM**
*/
#define Msg_WaterObject 1022
struct msg_waterobject {
  int    msg_len;       /* length of this message (excluding msglen) */
  int    msg_type;      /* msg_groundobject */
  int    type_index;    /* common type index (from data file). */
  int    generic_index; /* the generic object index. */
  float  easting;       /* meters easting from southwest corner. */
  float  northing;      /* meters northing from southwest corner. */
  float  heading;       /* course (0 degrees is straight north) degrees */
  float  velocity;      /* velocity in kph */
  short  status;        /* status (see below). */
  short  flags;         /* object flags*/
};
```

```
/* LAST OF JANUS OBJECTS SENT SYNCHRONIZATION.
  Direction: JANUS -> WM
  Purpose: To indicate that we've sent the last of the JANUS objects to
   the WM. Note that this goes into the "msg_sync" structure.
  When: **Janus initializes WM**
#define Msg_Sync_Janus_Objects 1001
/* MIRROR OBJECT INITIALIZATION
  Direction: WM -> JANUS
  Purpose: This is a WM ground object controlled by DIS. For the time being, there isn't
stuff in here about mine clearing and the like. This is the JANUS coordinate system (i.e.
remove the offset and convert to kilometers).
  When: **WM initializes JANUS**
#define Msg_DIS_GroundObject 1050
struct msg_DIS_GroundObject {
 int    msg_len;       /* Length of this message (excludoing msg_len) */
 int    msg_type;      /* Msg_DIS_GroundObject */
 int    type_index;    /* Common type index (from data file). */
 int    generic_index; /* The generic object index. */
 int    side;          /* What side is it on? =1 blue, =2 red. */
 int    mounted;       /* Mounted or not. */
 int    mopp;          /* =1 if in protective clothes. */
 float  easting;       /* Easting in kilometers */
 float  northing;      /* Northing in kilometers. */
 float  dirview;       /* Direction of view. */
 float  orientation;   /* Direction pointed. */
};
#define Msg_DIS_AirObject 1051
struct msg_DIS_AirObject {
 int    msg_len;       /* Length of this message (excluding msg_len) */
 int    msg_type;      /* Msg_DIS_GroundObject */
 int    type_index;    /* Common type index (from data file). */
 int    generic_index; /* The generic object index. */
 int    side;          /* What side is it on? =1 blue, =2 red. */
 int    mounted;       /* Mounted or not. */
 float  easting;       /* Easting in kilometers */
 float  northing;      /* Northing in kilometers. */
 float  dirview;       /* Direction of view. */
 float  orientation;   /* Direction pointed. */
 float  agl;           /* Above ground level (meters). */
 float  roll;          /* Roll (radians). */
 float  pitch;         /* Pitch */
 float  yaw;           /* yaw */
};
```

```
/* Future definitions. */
#define Msg_DIS_WaterObject 1052

/* JANUS GROUND OBJECT PLAN
  Direction: WM -> JANUS
  Purpose: JANUS object path for ground objects. These are sent during
    initialization for all objects with plans. Objects without plans are
    assumed to be stationary. We send all NUMNODES each time.
*/
#define Msg_Ground_Plan 1053
struct msg_ground_plan {
 int    msg_len;          /* Length of message (excluding msglen) */
 int    msg_type;          /* Msg_GroundObject */
 int    type_index;        /* Common type index (from data file). */
 int    generic_index;      /* The generic object index. */
 int    speed_type;        /* Sprint or normal. */
 int    start_index;       /* Where does this start in previous path? */
 float  nodex[NUMNODES];     /* X coordinates. */
 float  nodey[NUMNODES];     /* Y coordinates. */
 short int timed[NUMNODES];   /* Time at node. */
 unsigned char flags[NUMNODES]; /* Various flags. */
};

/* Types for the speed_type field above. */
#define SPEED_NORMAL 0
#define SPEED_SPRINT 1

/* END OF DIS OBJECT INITIALIZATION
  Direction: WM -> JANUS
  Purpose: This message is sent when the WM finishes sending objects
   to JANUS.
  When: **WM initializes JANUS**
*/
#define Msg_Sync_WM_Objects 1002


/* JANUS IS READY TO GO!
  Direction: JANUS -> WM
  Purpose: JANUS is really done initializing and is ready to get
    synchronized with the real world.
  When: just after **WM initializaes JANUS**
*/
#define Msg_Sync_GO 1003
```

```
/*****************************************************************************
                    JANUS <-> WM EXECUTION

  Messages exchanged during normal operations, once an entity has been
   defined.

*****************************************************************************/
/* END OF JANUS UPDATE SET.
   Direction: JANUS -> WM
   Purpose: Janus sends this message when it is done with it's updates.
     At this time the WM is free to start sending it's updates.
   When: **Janus upates WM**
*/
#define Msg_Sync_JANUS_Updates  1004

/* END OF WM UPDATE SET.
   Direction: WM -> JANUS
   Purpose: This is sent when the WM is done with it's updates to Janus. This kicks off the
next Janus simulation cycle. We also get the clock time so that Janus can decide what to do
if its lagging behind.
   When: **WM updates JANUS**
#define Msg_Sync_WM_Updates  1005
struct msg_sync_wm_updates {
  int    msg_len;       /* Length of this message (excluding msglen) */
  int    msg_type;      /* Msg_GroundObject */
  float  clock;         /* The real time clock (in seconds). */
};
/* GROUNDOBJECT POSITION REPORT
   Direction: JANUS <--> WM
   Purpose: Works in both directions to
#define Msg_Ground_Update 1030
struct msg_ground_update {
  int    msg_len;       /* Length of this message (excluding msglen) */
  int    msg_type;      /* Msg_GroundObject */
  int    generic_index; /* The generic object index. */
  float  time;          /* Time in minutes since start of simulation*/
  float  easting;       /* Meters easting from southwest corner. */
  float  northing;      /* Meters northing from southwest corner. */
  float  heading;       /* Course (in radians, east is 0.0) */
  float  velocity;      /* Velocity in KPH */
  short  status;        /* Status (see below). */
  short  flags;         /* Object flags*/
};
```

```
/* AIROBJ Update MESSAGE - tell the WM about a type of air object. */
#define Msg_Air_Update 1031
struct msg_air_update {
 int    msg_len;      /* Length of this message (excluding msglen) */
 int    msg_type;     /* Msg_GroundObject */
 int    generic_index; /* The generic object index. */
 float  time;         /* Time in minutes since start of simulation*/
 float  easting;      /* Meters easting from southwest corner. */
 float  northing;     /* Meters northing from southwest corner. */
 float  heading;      /* Course (0 degrees is straight north) degrees */
 float  velocity;     /* Velocity in KPH */
 float  agl;          /* Above ground level. */
 float  roll;         /* Roll of the Object*/
 float  pitch;        /* Pitch of the Object*/
 float  yaw;          /* Yaw of the Object*/
 short  status;       /* Status (see below). */
 short  flags;        /* Object Flags*/
};


/* Update Ships and floating stuff*/
#define Msg_Water_Update 1032
struct msg_water_update {
 int    msg_len;      /* Length of this message (excluding msglen) */
 int    msg_type;     /* Msg_GroundObject */
 int    generic_index; /* The generic object index. */
 float  time;         /* Time in minutes since start of simulation*/
 float  easting;      /* Meters easting from southwest corner. */
 float  northing;     /* Meters northing from southwest corner. */
 float  heading;      /* Course (0 degrees is straight north) degrees */
 float  velocity;     /* Velocity in KPH */
 short  status;       /* Status (see below). */
 short  flags;        /* Object Flags*/
};
```

```
/*****************************************************************************
                JANUS <-> WM Direct Fire
 These go both directions for direct fire events. There is both a firing
 event and splash.

*****************************************************************************/

/* Direct Fire event: Currently this goes in both directions. Later on, we need to add a
return message for a DIS object kill (or not kill) of a JANUS object. */
#define Msg_Direct_Fire 1040
struct msg_direct_fire {
 int    msg_len;       /* Length of this message (excluding msglen) */
 int    msg_type;      /* Msg_GroundObject */
 int    firer;         /* Who Shot*/
 int    target;        /* Who was the target */
 int    munition_type; /* What was shot */
 float  f_easting;     /* Firer Meters easting from southwest corner. */
 float  f_northing;    /* Firer Meters northing from southwest corner. */
 float  t_easting;     /* Impact Meters easting from southwest corner. */
 float  t_northing;    /* Impact Meters northing from southwest corner. */
 float  clock;
 short  status;        /* Status (see below). */
 short  flags;         /* Object Flags*/
};
/* Something impacted (could be a munition, could be a flying cow!) */
#define Msg_Impact 1042
struct msg_impact {
 int    msg_len;       /* Length of this message (excluding msglen) */
 int    msg_type;      /* Msg_GroundObject */
 int    firer;         /* Who shot. */
 int    firedon;       /* Who got shot */
 int    munition;      /* Common type index (if any) of thing fired.
                          Only used for throwing cows! */
 int    firesequence;  /* Fire sequence identification. */
 int    munition_type; /* What was shot */
 float  f_easting;     /* Firing location. */
 float  f_northing;
 float  t_easting;     /* Impact meters easting from southwest corner. */
 float  t_northing;    /* Impact meters northing from southwest corner. */
 float  clock;         /* When it should happen. */
 short  status;        /* Status (see below). */
 short  flags;         /* Object Flags */
};
```

```
/*************************************************************************
                JANUS <-> WM Indirect Fire
 These go both directions for direct fire events. There is both a firing
 event and splash. An indirect can have a kill or a proximity impact.

 *************************************************************************/

/* Indirect fire (arty) */
#define Msg_Indirect_Fire 1041
struct msg_indirect_fire {
  int    msg_len;       /* Length of this message (excluding msglen) */
  int    msg_type;      /* Msg_GroundObject */
  int    firer;         /* Who Shot */
  int    munition;      /* Common type index (if any) of thing fired.
                           Only used for throwing cows! */
  int    firesequence;  /* Fire sequence identification. */
  int    munition_type; /* What was shot */
  float  f_easting;     /* Firer Meters easting from southwest corner. */
  float  f_northing;    /* Firer Meters northing from southwest corner. */
  float  t_easting;     /* Target Meters easting from southwest corner. */
  float  t_northing;    /* Target Meters northing from southwest corner. */
  float  clock;         /* Time this happens. */
};

/* Something impacted but we don't know on who(m), just where. Let the
  targets figure it out. */
#define Msg_Proximity_Impact 1043
struct msg_proximity_impact {
  int    msg_len;       /* Length of this message (excluding msglen) */
  int    msg_type;      /* Msg_GroundObject */
  int    firer;         /* Who shot. */
  int    munition;      /* Common type index (if any) of thing fired.
                           Only used for throwing cows! */
  int    firesequence;  /* Fire sequence identification. */
  int    munition_type; /* What was shot */
  float  f_easting;     /* Firing location. */
  float  f_northing;
  float  t_easting;     /* Impact meters easting from southwest corner. */
  float  t_northing;    /* Impact meters northing from southwest corner. */
  float  clock;         /* When it should happen. */
  int    flags;         /* =0 ground impact, =1 close call. */
};
```

```
/* Something impacted and we know who it impacted on and where. */
#define Msg_Kill_Impact 1044
struct msg_kill_impact {
 int    msg_len;       /* Length of this message (excluding msglen) */
 int    msg_type;      /* Msg_GroundObject */
 int    firer;         /* Who shot. */
 int    target;        /* Who (may have) died. */
 int    munition;      /* Common type index (if any) of thing fired. */
 int    firesequence;  /* Fire sequence identification. */
 int    munition_type; /* What was shot */
 float  f_easting;     /* Firing location. */
 float  f_northing;
 float  t_easting;     /* Impact meters easting from southwest corner. */
 float  t_northing;    /* Impact meters northing from southwest corner. */
 float  clock;         /* When it should happen. */
 short  status;        /* Status (see below). */
 short  flags;         /* Object Flags*/
};

/
**********************************************************************

                WM -> JANUS Misc. Messages.

**********************************************************************/

/* We have a correction to JANUS position for a JANUS vehicle. */
#define Msg_Fix_Janus_Position 1060
struct msg_fix_janus_position {
 int    msg_len;       /* Length of this message (excluding msglen) */
 int    msg_type;      /* Msg_Fix_Janus_Position */
 int    generic_index; /* Generic index of object. */
 float  new_easting;   /* New position. */
 float  new_northing;  /* New northing. */
 float  new_velocity;  /* New velocity. */
 float  new_dirview;   /* New direction of view. */
 short  flags;         /* See flags below. */
};

/* If the fix should causes Janus to skip the the next segment, use that flag. */
```

```
#define SAME_SEGMENT 0
#define NEXT_SEGMENT 1


/
**************************************************************************

                    Public Status


**************************************************************************/

/* Public status values that everyone should know about and use.  Please fix
   the names in status.c and the limits below when adding to this list. */
#define STATUS_DEAD -1
#define STATUS_HULK -2
#define STATUS_ONGROUND 1
#define STATUS_MOUNTED 2
#define STATUS_STOPPED 3
#define STATUS_ALIVE 4
#define STATUS_INACTIVE 5
#define STATUS_AIRBORNE 6

#define STATUSMIN -2
#define STATUSMAX 6

/*flags */
#define ORIENT_VALID 0x0001
#define FLAGS_CLEAR  0x0000
#define SUBMERGED    0x0002
#define TARGETVALID  0x0004

/
**************************************************************************

                    End of the simulation


**************************************************************************/

/*After we are all done, we need to be able to exit the program gracefully*/
#define  Msg_Sync_Done  1009
```

LIST OF REFERENCES

[A2ATD] U.S. Army Material Systems Analysis Activity, *Anti-Armor Advanced Technology Demonstration (A2 ATD), Experiments 2, 3, 4 and 5, Independent Evaluation Plan and Test Design Plan*, Draft, Aberdeen Proving Ground, MD, February 1994.

[ARMY93] Department of the Army, *Army Focus 1993*, US Army Publication and Printing Command, The Pentagon, Washington DC, September 1993.

[ARMY95] Department of the Army, *United States Posture Statement FY95: Challenges and Opportunities*, Office of the Chief of Staff, United States Army, The Pentagon, Washington DC, February 1994.

[IST93a] Institute for Simulation and Training, *Standard for Information Technology - Protocols for Distributive Interactive Simulation Applications*, Draft 2.0.3, Orlando, FL, May 1993.

[IST93b] Institute for Simulation and Training, *Enumeration and Bit Encoded Values for Use with Protocols for Distributive Interactive Simulation Applications*, Orlando, FL, June 1992.

[JANU93] Department of Army, *The Janus 3.X/UNIX Model User's Manual*, Headquarters TRADOC Analysis Center, ATRC-ZD, Ft. Leavenworth, KS, May 1993.

[JANU93a] Department of Army, *The Janus 3.X/UNIX Model Data Manager's Manual*, Headquarters TRADOC Analysis Center, ATRC-ZD, Ft. Leavenworth, KS, May 1993.

[JANU93b] Department of Army, *The Janus 3.X/UNIX Model Computer System Operator's Manual*, Headquarters TRADOC Analysis Center, ATRC-ZD, Ft. Leavenworth, KS, May 1993.

[JANU93c] Department of Army, *The Janus 3.X/UNIX Model System Design Manual*, Headquarters TRADOC Analysis Center, ATRC-ZD, Ft. Leavenworth, KS, May 1993.

[JANU93d] Department of Army, *The Janus 3.X/UNIX Model Software Design Manual*, Headquarters TRADOC Analysis Center, ATRC-ZD, Ft. Leavenworth, KS, May 1993.

[JANU93e] Department of Army, *The Janus 3.X/UNIX Model Software Programer's Manual*, Headquarters TRADOC Analysis Center, ATRC-ZD, Ft. Leavenworth, KS, May 1993.

[JOHNS] Johnson, Carol N., Stich, Angela, Vye, Patrick, *Integration of NVL-2D into JANUS*, Draft, RAND Arroyo Center, Santa Monica, CA, August 1994.

[KARR] Karr, Clark R. and Root, Eric, Integrating Aggregate and Vehicle Level Simulations, "*Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*", published by the Institute for Simulation and Training, Orlando, FL, May 1994.

[Locke94] Locke, John, *An Introduction to the Internet Networking Environment and SIMNET/DIS*, Computer Science Department, Naval Postgraduate School, Monterey, CA, August 1994.

[MACED] Macedonia, Michael R., Zyda, Michael J., Pratt, David R., Barham, Paul T., Zeswitz, Steven, *NPSNET: A Network Software Architecture for Large Scale Virtual Environments*, Computer Science Department, Naval Postgraduate School, Monterey, CA, June 1994.

[MARTI] Marti, Jed, *JLINK Integrating JANUS and BDS-D Project Overview*, Draft, RAND Arroyo Center, Santa Monica, CA, August 1994.

[McDon93] McDonough, James G., Doorways to the Virtual Battlefield, Proceedings from "*The First West Coast EFDPMA Conference and Exhibition on Virtual Reality: The Commitment to Develop VR*", Illusion Engineering, Inc., Westlake Village, CA, December 1993.

[Milton] Milton, Fenner, *Memorandum for Deputy Undersectretary of the Army (Operational Research), Subject: JANUS/BDS-D Interface Support for Advanced Land Combat*, Office of the Director of Defence Research and Engineering, Washington DC, December 1992.

[OSBO91] Osborne, William D., *NPSNET: An Accurate Low-Cost Technique for Real-Time Display of Transient Events: Vehicle Collisions, Explosions and Terrain Modifications*, Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1991.

[PRAT93] Pratt, David R., *A Software Architecture for the Construction and Management of Real-Time Virtual Worlds*, Dissertation, Naval Postgraduate School, Monterey, June 1993.

[PRAT93a] Pratt, David R., Lessons Learned from NPSNET, Proceedings from "*The First West Coast EFDPMA Conference and Exhibition on Virtual Reality: The Commitment to Develop VR*", Illusion Engineering, Inc., Westlake Village, CA, December 1993.

[PRATT94] Pratt, David R., Johnson, Matthew A., Locke, John, The Janus/BDS-D Linkage Project: Constructive and Virtual Model Interconnection, "*Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*", published by the Institute for Simulation and Training, Orlando, FL, May 1994.

[SMIT93] Smith, Richard Samuel, *NPSNET: Scripting of the Three-Dimensional Interactive Systems for use in the JANUS Combat Simulation,* Naval Postgraduate School, Monterey, CA, September 1993.

[WALT92] Walter, Jon C., and Warren, Patrick T., *NPSNET: Master's Thesis in Computer Science, JANUS-3D Providing Three-Dimensional Displays for a Traditional Combat Model*, Naval Postgraduate School, Monterey,CA, September 1992.

[ZOBRIS] Zobrist, Albert L., *Integration of JANUS to BDS-D: Making Detection Consistent*, Draft, RAND Arroyo Center, Santa Monica, CA, August 1994.

[ZYDA92] Zyda, Michael J, Pratt, David R, Monahan, Gregory, and Wilson, Kalin P., *NPSNET: Constructing a 3D Virtual World*, Symposium on 3D Graphics, '92 Proceedings, April 1992.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center — 2
   Cameron Station
   Alexandria, VA 22304-6145

2. Dudley Knox Library — 2
   Code 052
   Naval Postgraduate School
   Monterey, CA 93943

3. Chairman Ted Lewis, Code CS/Lt — 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

4. Professor D. R. Pratt, Code CS/Pr — 5
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

5. Professor G.M. Lundy, Code CS/Ln — 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

6. Dr. Jed Marti — 1
   RAND, The Arroyo Center
   1700 Main Street P.O. Box 2138
   Santa Monica, CA 90407-2138

7. Will Brooks — 1
   AMSAA
   AMXST-GA
   APG, MD 21028

8. CPT Jude Fernan — 2
   TRAC-MTRY
   P.O. Box 8692
   Monterey, CA 93943

9. Chris Christenson
Institute For Defense Analysis
1801 N. Beauregard St.
Alexandria, VA 72311 — 1

10. Bill Caldwell
R&A
500 Sloat Ave
Monterey, CA 93940 — 1

11. LTC Michael Proctor
TRAC-MTRY
P.O. Box 8692
Monterey, CA 93943 — 1

12 Paul Barham
Computer Science Department
Naval Postgraduate School
Monterey CA, 93943 — 1

13. CPT Matthew A. Johnson
Department of EE&CS
West Point, NY 10996 — 3